**Jenkins Multibranch Pipeline**

* Let’s create a new public repository there, and save the repository URL because we’ll use it later.
* Head over to your Jenkins instance and create a new item. Enter a name for the job, and select the “Multibranch Pipeline” option at the end of the screen. Then, click on the OK button.
* In the next screen, go to the “**Branch sources**” tab, click on the “**Add source**” button, and choose “Git” from the dropdown menu. Then enter the repository URL. Because this is a public repository, we don’t need to configure any credentials.
* We’ll use all the defaults for now. So, scroll down to the end of the screen and click on the “**Save**” button to finish creating the job. In the logs, you should see that Jenkins is cloning the repository and it creates the job successfully, even if it’s an empty repository.
* Let’s create a Jenkinsfile in the root directory, and for now let’s keep it simple, like this:

pipeline{

agent any

stages{

stage('GetCode'){

steps{

git branch: 'Release', url: 'https://github.com/skatta3/UserManagement.git'

}

}

stage('Build'){

steps{

sh 'mvn clean package'

}

}

}

}

* After you commit and push the Jenkinsfile to the remote repository, you should be able to see the file created in the main branch.
* Now, we need to ask Jenkins to scan the repository to find the new branch we just created. Head over to your Jenkins job, and click on “Scan Multibranch Pipeline Now” on the left side of the screen. Wait for a minute and refresh the screen. Now you should see that a new branch appears and a Jenkins job has been created automatically.
* At this point, you haven’t seen the benefit of creating a multibranch pipeline job. So, let’s create two more branches in git to see what happens in Jenkins. Open a terminal in your computer and clone the git repository you created before (in case you haven’t done so). Then, run the following commands to create two branches from the main branch:
  + git checkout -b fix-123
  + git checkout main
  + git checkout -b dev-456
* Now you need to push these branches to the remote repository so that the Jenkins job is able to see them and create the jobs automatically. To do so, run these commands:
  + git push --set-upstream origin dev-456
  + git push --set-upstream origin fix-123
* If you wanted to delete the branches in local and Remote

// delete branch locally

git branch -d localBranchName

// delete branch remotely

git push origin --delete remoteBranchName

* If you go to GitHub, you should see that the two new branches were pushed successfully.
* Let’s go back to your multibranch job in Jenkins. Click again on the “Scan Multibranch Pipeline Now” to discover the new branches and create the new jobs. Wait for a minute and refresh the page. You should see that two new jobs were created because of the new branches you just pushed.
* Those two new jobs have the same configuration as the one you created initially (the multibranch pipeline). Therefore, you can’t edit those jobs, at least not from the UI, but you can run them and view its logs and change the pipeline through the Jenkinsfile. This way, you can promote changes to other branches and test the pipeline several times before you push it to the main branch. Let’s see how this would work by making a change in the job through git branches.

---------------------------------------------------------------------------------------------------------------------

**Step 1**: Open Jenkins home page (http://localhost:8080 in local) & click on New Item from the left side menu.

**Step 2:** Enter **Jenkins job name** & choose the style as **multibranch pipeline** and click **OK**.

**Step 3**: In the **configure** page, we need to configure only one thing – **The Git Repo source**.

Scroll down to **Branch Sources** section & click on **Add Source** dropdown.

Choose **Git** as Source as our Sample GitHub repo is hosted there.

**Step 4**: Enter the **Repository HTTPS URL** as https://github.com/iamvickyav/spring-boot-h2-war-tomcat.git

Since our GitHub repo is hosted as a public repo, we don’t need to configure credentials to access it. For enterprise/private repos, we may need credentials to access them.

**Step 5**: Leave the rest of the configuration sections as such for now and click on the **Save** button at the bottom.

On saving, Jenkins will perform the following steps automatically.

**Scan Repository Step**

* Scan the Git repo we configured
* Look for the list of branches available in the Git repo
* Select branches which has Jenkinsfile

**Running Build Step**

* Run build for each of the branches found in previous step with steps mentioned in Jenkinsfile

From the Scan Repository Log section, we can understand what happened during the Scan repository step

Since we have an only a **master branch** in our git repo, Scan Repository Log says **1 branches were processed**

After the scan is complete, Jenkins will create & run a build job for each processed branch separately.

In our case, we had only one branch called master. Hence build will run for master branch alone. We can check the same by clicking on **Status** in the left side menu.

We can see a build job created for the master branch in the status section.